Artificial Intelligence and Neural Networks: Steps Toward Principled Integration

Leonard Uhr
Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

Vasant Honavar*
Department of Computer Science
Iowa State University
Ames, IA 50011

June 16, 1994

1 Introduction

The attempt to understand intelligence entails building theories and models of minds and brains, both natural and possible. From the earliest writings of India and Greece, this has been a central problem for philosophers and psychologists. Among the first uses of the modern electronic computer, in the early 1950s, was the development of computer programs that perceive and reason, and model such intelligent processes in humans. The rapid growth through the 1960s of research on these problems was variously called artificial intelligence, bionics, and cybernetics among other things.

A variety of problems were attacked, including recognizing machine or handprinted letters or numbers, proving theorems in geometry or logic, answering questions about baseball or the parts of the body, playing checkers or chess, and learning. A variety of approaches were taken, including:

- Searching for paths to solutions from sets of given facts and background knowledge (e.g., the axioms of a logical system, the initial board for a game);
- Extracting features from images and combining and using them to recognize objects;
- Parsing or finding and using keywords in questions to infer plausible answers;
- Storing and retrieving information in internal networks of memories that recorded, and / or processors that computed, this information; and

*This work was partially supported by Iowa State University College of Liberal Arts and Sciences.

- Learning - including trying to infer what should be learned and store that in usable form, induce concept classes from individual examples, generalize, or/and build and modify the internal network.

Most people wrote programs that they executed on serial stored program computers (or von Neumann machines) that have dominated information processing from the 1950s on. Some had more parallel brain-like networks in mind, and wrote more or less precise specifications of what such a realization of their programs might look like. Some wrote programs or specifications that used networks of neuron-like primitive units (almost always simulating these on von Neumann computers). A few actually built hardware networks of units (some neuron-like, some logic-based); but everybody agreed that this was a matter of convenience since anything such a network was capable of doing could therefore be done by a program for any general-purpose serial computer - and it was usually far easier to write and debug a program than to actually build the hardware computer).

It is often suggested that two major approaches have emerged - symbolic artificial intelligence (SAI) and (numerical) artificial neural nets (ANN), and some have even suggested that they are fundamentally different. But one of the few things we know with absolute certainty is that all the different approaches, and all the different types of languages, computers, and networks used to implement information-processing systems are fundamentally equivalent. This is probably best explained by introducing the extremely important theoretical developments that underlie the actual design, construction, programming, and running of computers.

2 The Exact Equivalence of All General-Purpose Systems, Including Languages, Computers, and Networks

Much of the theoretical foundations of computer science can be traced at least as far back as Alan Turing's encounter (as a student of mathematical logician M. H. A. Newman at Cambridge in the mid 1930s) with Hilbert's decision problem the essence of which is to ask if there exists an effective procedure for determining whether or not a certain conclusion logically follows from a given set of axioms. The formal notion of an algorithm as we understand it today is the result primarily of Turing's attempt to formalize the intuitive notion of an effective procedure. Essentially, an algorithm is a set of rules that precisely specify what to do under any given set of circumstances e.g., "2+2 = 4" or "$area = length \times width$", etc. To reflect the mechanical nature of what is involved in carrying out the steps of an algorithm, Turing invented a hypothetical computer now called the Turing machine and used it to formalize Hilbert's decision problem. Essentially, a Turing machine has a processor *head* that reads (and erases) symbols from and writes symbols to a potentially infinite memory *tape*that is divided into squares each of which is capable of storing a symbol. The behavior of a Turing machine is governed by an algorithm which is realized in terms of what we now call a program. A program is a finite set of instructions selected from a *general-purpose* set of simple instructions. A necessary and sufficient set of instructions is simply: 'fetch (from the tape)', 'store (onto the tape)', 'move the head one square to the left ', 'move the head one square to the right' and 'branch conditionally (if the current square contains a 0, do a, else do b)'.

A "universal" Turing machine" - one that can do anything that any other Turing machine *could possibly do* - can be constructed simply by storing on the tape a simple program of instructions that will interpret and instruct the head how to execute any program that has been stored on the tape and is written in the language that interpreter program defines. So long as the tape is large enough to handle all the necessary data (including input, intermediate results, and the actual programs that instruct the machine what to do), such a universal system can do anything that can be described. The *potentially infinite tape* (which serves as the system's memory) is simply a way to ensure the system won't run out of space. Turing proved that any parallel or/and serial structure of describable processes (that is, processes that could be stated in the form of input-output functions) can be executed by such a machine.

Turing and others also showed that there could be many different versions of such a universal information-processing/computing system, including ones with 2, 3, or any number of heads and/or memories, all of which are equivalent to the universal Turing machine. About the same time, Alonzo Church and his students Steven Kleene and Barkley Rosser at Princeton had begun to investigate several frameworks for computation including the lambda-calculus and recursive functions. Still other formulations were made, including McCulloch-Pitts neural networks, Markov algorithms, Petri nets, and two systems by Emil Post - an independent invention of the Turing machine and Post productions. In addition, all of these (given potentially infinite memory) were *proved* to be *exactly* equivalent to the Turing machine. That is, any structure of parallel and/or serial processes that could be executed by a Turing machine could in principle be executed by any of these other systems.

There are hundreds of different general-purpose programming languages, but - just as with computers - they are all general-purpose: Anything that can be programmed (that is, that is describable) can be programmed in any language.

Since any computer can execute any structure of processes, a program can be written for any computer that translates (sometimes called a compiles or interprets) any programming language into that computer's machine language. Almost all computers have translators for at least one easier-to-use "higher-level language." But the more important point is that a translator from any language to any computer's machine language, or for that matter to any other general-purpose language, can always be built. And any general-purpose language or computer can do anything that any other can. (The same structure of processes may be millions of times faster on one computer than another, so from any practical point of view that may not be feasible But eventually, given enough time, the results will be the same.)

Turing also proved that there are limits to what can be done with computers, since there are problems where the system can never know before finding a solution (e.g., proof of a theorem in first order logic) whether there exists one and hence may never halt. Some people have used these limits to argue that computers therefore cannot possibly ever be intelligent in all the ways that human beings are. It is important to make clear that this is *not* an issue of whether computers can think, or can possibly ever be made to behave intelligently. Rather, whatever it is that a general-purpose information processor can do can *therefore* be done by writing a program in any language for any other computer.

3 Different (But Equivalent) Theoretical Formulations Have Strongly Conditioned Implemented Languages, Computers, and Systems

The design of computers and programming languages often grew out of, and was strongly influenced by, one or another of these theoretical formulations. Thus the traditional serial stored-program computers and their basic machine languages have much the flavor of a single-head Turing machine. Turing was invited to Princeton by Alonzo Church in 1936. Since von Neumann and Turing talked frequently during that time, it seems likely that von Neumann's thinking on stored-program computers was strongly influenced by Turing's work. Lisp is based on the lambda calculus and recursive functions. Comit, Snobol, and expert production systems have origins in Post productions. Neural (connectionist) networks are derivatives of McCulloch-Pitts networks.

A number of things follow from this history and the exact equivalence among all general-purpose languages and general-purpose computers that throw important light on current research and the associated controversies in Artificial Intelligence and Neural (Connectionist) Networks.

First, although some people have asserted that traditional serial symbol list-processing Artificial Intelligence systems and numerical parallel Artificial Neural Nets are fundamentally and irreconcilably different, they cannot possibly be. For they are always executed on some computer or another, typically using a program written in a general-purpose language (Lisp, C, etc.).

Second, although some people assert that SAI apply complex serial processes to lists of symbols, while NANN use simple parallel processes on numbers, this reflects the basic structures and primitive operations each is given and the preconceptions of the people who use them. Obviously, if one can work with symbols, or numbers, or lists, or whatever, therefore so can the other. This gives absolute assurance that each can handle everything, and great hope that the necessary methods will be found once we put both in a common framework.

Very unfortunately, the 1960s saw the field separate into two major streams. Rather than continue to explore alternative approaches, continually comparing results and learning from one another, the situation degenerated into two not only separate but perhaps even mutually hostile camps. By far the larger was the group developing traditional SAI. The dominant paradigm was that of serial search (as executed by a serial program coded in a list-processing language, usually Lisp, on a serial computer) using complex processes (algorithms when available, but usually heuristics) and data structures (usually lists of symbols or logical expressions). In contrast, the smaller group was developing parallel structures of very simple neuron-like processors mostly performing numerical operations (but occasionally transformations over symbol structures as well). In addition, a small number of researchers (e.g., those working on problems in computer vision) followed the middle ground.

Traditional SAI pretty much swamped things, with jazzy demonstrations, impressive claims of success, and spectacular predictions of imminent progress. Minsky and Papert's systematic study of one of the earliest neural network models – the perceptrons, explained some of the limitations of the the prevalent neural network models of the time. Misleading interpretations of the import of proofs about the limitations of perceptrons led many people - too many - to wrongly conclude that NANN could not possibly handle problems that SAI could handle. Since that could not possibly be the case, since they are both general-purpose and even implemented in the same programming languages running on the same computers, in retrospect, it is hard to understand how such assertions could be taken seriously. But they effectively stifled much of the research on NANN - and even discouraged much of the work that was going on in the middle - for the next 10-15 years.

Starting in the early 1980s, there has been an almost explosive increase in interest in NANN. At the same time, some people feel that SAI has stagnated although in reality the over-optimism and excessive hype have probably obscured and soured significant although far more modest successes in many important areas. This has led to claims from certain quarters that NANN were not only achieving striking successes (reminescent of the earlier SAI hype), but were handling problems on which SAI had failed and could never possibly handle. And the great (more perceived than real) separation between the two approaches tends to limit potentially very beneficial cross-fertilization of ideas towards the solution of difficult problems in artificial intelligence and cognitive modelling.

This book is dedicated to pulling the SAI and NANN approaches closer, if not back, together. That may end up taking a long time. But it is very nice to know, as the above examinations guarantee that we do, that not only can they be pulled together but ultimately they are the same – At the heart of both approaches is the very same working hypothesis – that cognition, at some level, can be modeled by computation. Our problems lie first in dispelling misinformed and wrong notions, and second (much more difficult) in developing systems that take advantage of both paradigms to build robust, flexible, and versatile intelligent systems.

4 What, Really, Are SAI and NANN?

At first glance, SAI and NANN do indeed look radically, if not fundamentally, different. Compare the following prototypical SAI and NANN systems.

The SAI system is given a set of rules for moving from one expression (a string of symbols) to another, and asked to find a path between two expressions (e.g., the initial state and the goal, the puzzle and its solution, the axioms and a theorem to be proved, etc.). In any non-trivial problem a blind exhaustive search for a path will be impossibly slow, and there will be no known algorithm - a procedure for directly computing that path.

The NANN system is given a set of nodes (usually arranged in several layers, from the layer that receives inputs to the layer that outputs), and asked to output the correct response to each input it receives. The input-output behavior of the network is a function of the network architecture, the functions computed by the individual nodes and parameters such as the weights. For example, the solution of an optimization problem (traditionally solved using search) can be formulated as a problem of locating a state of a suitably designed network that corresponds to one of minimum energy (which is defined to correspond in some natural way to the optimality of the solution being sought). The network dynamics are set up so as to accomplish this ideally without additional explicit control.

Some networks are also equipped with learning mechanisms which enable them to potentially learn the desired input-output behavior by changing its internals - today almost always by changing the weights.

For example, SAI systems handle games like chess as follows: The initial board is established as the given, and a procedure is coded to compute whether a win-state has been achieved. In addition, procedures are coded to execute legal moves and (usually) to compute heuristic assessments of the promise of each possible move, and to combine the separate heuristic assessments into an overall value that will be used to choose the next move. Finally, all these are put into a total structure that applies the appropriate heuristics, combines their result and evaluates alternative moves, and actually makes a move, then waits for and senses the opponent's moves, uses it to update the board (probably checking that it is indeed a legal move), and loops back to make its own next move. (For simplicity the lookahead with minimax that most game-players use has been ignored, but that is essentially more of the same.)

The typical multi-layer neural net will in theory learn to play chess as well as the best possible SAI system of this sort. But in practice, unless a lot of prior structure is built into the network architecture, it almost certainly would not be that much different from the proverbial monkeys at typewriters.

A typical problem for an NANN is one where a pattern (e.g., a binary image) is provided as input and the system has to produce the appropriate output (e.g., name or class to which the pattern belongs). Indeed, carefully designed neural networks have been successfully trained to solve many such pattern classification problems (e.g., low-level vision).

5 Toward a Principled Integration of SAI and NANN

It must be clear from the discussion in the previous section that at least on the surface it looks like SAI and NANN are each appropriate, and possibly even necessary for certain problems, and grossly inappropriate, if not impossible, for others. But of course each can do anything that the other can. The issues are ones of power, efficiency and elegance, not capabilities. Although outrageously and impossibly awkward or slow, there will be ultimate success.

This is a common problem in computing. One computer or programming language may be extremely well suited for some functions but awful for others, while a second computer or language may be the opposite. It suggests several possibilities, including:

1. Try to re-formulate and re-code the problem to better fit the computer or language.

2. Use one computer or language for some parts of the process and the other for others.

3. Build a new computer or language that contains constructs from each, and use these as appropriate.

4. Try to find as elegant as possible a set of primitives that underlie both computers or languages, and use these to build a new system.

The term *hybrid* is beginning to be used for systems that in some way try to combine SAI and NANN. If any of the above is called a hybrid probably and of the others should also. But usually hybrid refers to systems of type [2] or [3]. [3] and [4] would appear to be better than [2] (although harder to realize), since they would probably be more efficient and more elegant. Thus the capabilities of both SAI and NANN should be combined by tearing them apart to the essential components of their underlying processes and integrating these as closely as possible. Then the problem should be re-formulated and re-coded to fit this new system as well as possible. This restates a general principle most people are coming to agree on with respect to the design of multi-computer networks: the algorithm and the architecture should be designed to fit together as well as possible, giving algorithm-structured architectures and architecture-structured algorithms.

SAI and NANN each demonstrate at least one way of performing certain tasks naturally and thus pose the interesting problem for the other of doing something equivalent perhaps more elegantly, efficiently, or robustly than the other. This effort can give important insights toward solving the difficult problems of integrating the two approaches. These insights, plus the certain knowledge that the task can be accomplished - the living mind/brains are existence proofs of this fact - provide the impetus for such an effort.

6 Overview of the Book

This book is organized into five parts:

- The first part – *SAI versus NANN* is critical examination of the historical, philosophical and scientific origins, fundamental assumptions, and (perceived as well as real) differences and similarities among the two paradigms for artificial intelligence and cognitive modelling. It should be clear from the multiple perspectives presented here that their exact equivalence as computational models and their shared philosophical foundations suggest that there is much to be gained by their systematic synthesis within a common framework.

- The second part – *Representation and Inference* presents a careful analysis of important (and often controversial) issues in knowledge representation and inference that have played a major role in the unfortunate rift between SAI and NANN. The several examples of knowledge representation and inference structures demonstrate how SAI and NANN mechanisms can be combined within systems that can potentially take advantage of the strengths of both.

- The third part –*Perception* presents examples of systems that combine structures and processes drawn from both SAI and NANN paradigms to solve difficult problems in perception (e.g., visual pattern recognition).

- The fourth part – *Language* presents additional examples of integrated SAI and NANN systems that begin to attack problems in natural language that have hithertofore resisted solution.

- The fifth part –*Learning* critically examines SAI and NANN models of learning, identifies a number of areas which can potentially benefit from a careful synthesis of mechanisms drawn from both paradigms. It also presents some simple examples of such a synthesis.

6.1 SAI Versus NANN

Popular interpretations of SAI and NANN approaches often tend to be unnecessarily narrow and restrictive. It is therefore worthwhile examining in some depth how the two paradigms differ (if at all) from each other and what each of them has to offer to the enterprise of understanding and engineering intelligent behavior. Boden (chapter I), Chandrasekaran and Josephson (chapter II) and Oden (chapter V) examine this and related questions in detail.

Apart from the different (although equivalent) choice of the prefered model of computation, SAI and NANN each have benefited from and contributed to cognitive psychology (which has traditionally concerned itself with mental phenomena such as problem-solving) and neuroscience (with its emphasis on understanding the anatomy, physiology and neurochemistry of the brain) respectively. The last few years have seen a movement in cognitive science toward a multi-disciplinary approach toward the study of mind/brain. It is only natural that such an approach is likely to benefit from as well as contribute to a synthesis of SAI and NANN as paradigms for modelling mind/brain. McKenna (chapter IV) makes a case for an interdisciplinary approach involving neuroscience in the engineering of intelligent systems. Arbib (chapter III) offers an example of an evolving theoretical framework in the form of *schema theory* for the integration of SAI and NANN models.

6.2 Knowledge Representation and Inference

SAI and NANN systems differ significantly in the manner in which they represent and process knowledge. Knowledge in SAI systems is typically encoded in complex data structures (e.g., lists, trees, etc.) and procedures or rules. Computation involves typically explicit rule-driven manipulation of stored data strucures or serial chains of inference. The control and coordination of such inference chains is often rigid and centralized. However, it is unnecessary to limit SAI systems to serial rule-driven inference: several rules may be activated in parallel and can have interactive effects on the final outcome (as in parallel production systems). There is no compelling reason for using explicit rules in SAI systems; the rules may be directly implimented in the hardware that supports the computations involved and the choice is mostly a matter of convenience and efficiency. It is rather straightforward to map production rules into functionally equivalent neural networks.

Most NANN systems are characterized by massive fine-grained parallelism of computation. They are reminescent of data flow machines rather than stored program computers. They are typically designed to perform associative recall of patterns.It is often argued that NANN use distributed (and hence fault-tolerant) representation of knowledge (with each concept represented by patterns of activation over a population of neurons and each neuron contributing to the representation of many concepts). Some have claimed that NANN are incapable of performing deductive inference of the sort that is routinely used in SAI systems (for example, in theorem proving and expert production systems). Others have argued that NANN lack the ability to construct complex entities through a systematic composition of constituent components. Compositionality of representations is thought by many to be a prerequisite for higher cognitive abilities (e.g., language).

Given the exact equivalence of NANN and SAI as models of computation, it must be obvious that each can do whatever the other can. Most programming languages used to implement SAI systems have a built in capability for processing complex symbol structures. However symbol structures are not obvious parts of the NANN landscape – especially when NANN are built from very simple analog processors and there are no memory registers and synchronizing clocks. But this is simply a matter of implementing the necessary functional modules using neuron-like processors. For example, recurrent networks can be used to realize memory registers as well as clocks. The interesting questions have to do with whether particular NANN, SAI or hybrid realizations of knowledge representation and inference structures offer specific performance (speed, size, cost, robustness) advantages for particular tasks. The knowledge representation and inference mechanisms implemented in today's SAI and NANN systems are almost certainly a small subset of a much larger space of possibilities. Van Gelder and Port (chapter VI), Sharkey (chapter VIII), examine a variety of mechanisms for supporting compositionality of representations. Barnden (chapter VII) discusses possible NANN mechanisms of representation and inference of propositional attitudes (beliefs, intentions, etc.) that is a part of communication using natural language. Servan-Schreiber, Cleeremans, and McClelland (chapter XI) present an NANN implementation of (graded) state machines for learning temporal contigencies (e.g., a finite state grammar). Omlin and Giles (chapter XII) discuss the transformation of recurrent NANN representations of finite state grammars into rules and vice versa. Sun (chapter XIII) examines an NANN implementation of several essential functional modules for logical inference. Pinkas (chapter XIV) presents an architecture for logical inference that exploits a theoretical relationship between the proof procedure and energy minimization in a suitably designed NANN. Uhr (chapter XV) examines a number of NANN building blocks or functional modules (including those already used in digital and analog computers and those that appear to be found in brains) which can be used to build efficient and robust SAI, NANN, as well as hybrid systems.

Distributed representations and associative recall of patterns that are employed in many NANN models offer robustness in the presence of failure of components and noise. It is often argued that SAI systems are inherently less robust on account of their reliance on localized representations and rigid address-and-index-based access. But there is no reason to not implement associative recall or distributed storage and redundancy in SAI systems. Sharkey (chapter VIII) and Oden (chapter V) and MacLennan (chapter X) explore different aspects of distributed and localist representations. The mechanisms of distributed storage and recall and fault tolerance implemened in today's NANN and SAI systems are only suggestive of a much larger range of possibilities.

The knowledge representations in SAI systems typically use highly abstract symbolic descriptions. This in fact allows such systems to attack *high-level* cognitive tasks while circumventing the difficulties involved in arriving at the knowledge representations starting with raw sensory data. Since such symbolic representations are manipulated by purely syntactic means, it is often argued that they are devoid of intrinsic semantics (that is, meaning beyond that ascribed to them by the designer of the system). Harnad, Lubin and Hanson (chapter IX) propose mechanisms for semantic grounding of learned categories in analog representations. Such semantic grounding of symbolic representations is thought by many to be essential to replicating truly intelligent behavior.

6.3 Perception

The design of robust efficient and flexible systems that recognize complex patterns (e.g., three-dimensional objects from two-dimensional images) offer many challenges to both SAI and NANN systems. Most SAI systems used in computer vision tend to be rigid and inflexible in their use of representations and inference strategies. For example, programs that are designed to recognize faces or three-dimensional objects from two-dimensional images generally have difficulty with noise, partial occlusion, and variations in background illumination and the like. Furthermore, it is extremely difficult to program such systems with the knowledge necessary to perform adequately in realistic environments.

It is often claimed that NANN are especially good for performing tasks such as pattern classification. Yet in practice, the application of most such systems has been limited to tasks such as feature detection, signal processing, and recognition of two-dimensional patterns such as hand-written or printed characters. This is almost certainly due to their reliance on impoverished knowledge representation structures (e.g., bit vectors that fail to preserve much of the potentially useful neighborhood relations between adjacent pieces of a two-dimensional image and are generally incapable of efficiently representing useful invariances about the objects)and their lack (among other things) of adequate coordination and control mechanisms. It is becoming increasingly apparent that judicious use of domain-specific constraints on network architecture as well as the use of sufficiently expressive representational structures are essential for success in complex problems of machine perception. Kosslyn and jacobs (chapter XVI) explore a simple neurobiologically and psychophysically motivated mechanism for coordinating multiple representations of visual scenes for effective identification of both the category as well as the location of objects. Carpenter and Grossberg (chapter XVII) present a self-organizing NANN architecture that learns (in effect symbolic) production rules for pattern recognition and prediction. Their model illustrates successful emergence of goal-driven such as attention and novelty detection as properties of a suitably designed NANN architecture. Mjolsness (chapter XVIII) proposes a class of NANN implementations of stochastic grammars (of the sort that have been used in syntactic pattern recognition systems) for computer vision.

6.4 Language

The design of systems that learn and communicate in (at least restricted subsets of natural language) is perhaps among the most difficult challenges in artificial intelligence and cognitive modelling. The SAI approaches to this problem have tended to rely on postulation of (symbolic) language primitives and sets of rules of grammar for construction of more complex symbol structures (e.g., sentences) from the primitives. The problem of language learning, to the extent that it has been addressed, is viewed primarily as one of aquiring the rules of grammar. What is missing from such approaches is a satisfactory account of the acquisition of semantics or meanings associated with the symbol structures. As a result, over the recent years, many researchers have begun to explore models of language acquisition that are closely integrated with those of perception and motor behavior. Since NANN have been especially successful in modelling processes such as motion and shape detection, object location and motor control, problems in natural language offer a challenging testbed for the exploration of approaches that integrate both NANN and SAI. Miikkulainen (chapter XX) explores describes a modular NANN model of episodic memory for script-based story understanding. Dyer (chapter XIX) discusses a model of language acquisition designed to explore the representation and learning of basic concepts involving space, time, and change within a simplified simulated environment. Smolensky, Legendre and Miyata (chapter XXI) propose a new formalism which they call Harmonic Grammar as a framework for integration of both SAI and NANN mechanisms to provide a basis for a theory of language.

6.5 Learning

No computational account of cognitive processes can be complete without a detailed account of structures and processes that support learning. However, much of the early SAI work concentrated on the development of complex domain-specific, knowledge intensive programs (e.g., expert systems). The establishment of the needed knowledge-base in such systems is through a process of *knowledge engineering* which is a difficult and tedious task that entails making explicit the knowledge and working methods of the human expert which are often implicit and difficult to characterize in the form of precise rules that cover every possible scenario that the system has to deal with. The difficulties of knowledge engineering have in recent years stimulated the interest in learning programs. Learning in SAI systems involves additions and changes to the stored data structures and procedures. Many such systems are rigid and inflexible when they encounter noisy inputs. This limits their usefulness in many real-world applications (e.g., computer vision, speech recognition). In contrast, learning from examples is the prefered mode of knowledge acquisition in most NANN systems. Such learning typically involves small experience-induced changes to numerical parameters associated with the links and/or nodes in the network. NANN systems tend to have an advantage over SAI systems in noisy environments. But today's NANN systems are rather limited in their applicability to tasks such as learning to recognize and describe three-dimensional objects. Thus, the design of robust, flexible learning systems can benefit from a judicious integration of SAI and NANN mechanisms.

Goldfarb and Nigam (chapter XXII) point out the limitations of today's SAI and NANN systems for learning from examples and a model that they call an evolving transformation system which incorporates aspects of both. Shavlik (chapter XXIII) discusses a framework for incorporating rules of the sort used in SAI expert systems into NANN systems in ways that take advantage of both. Booker, Riolo and Holland (chapter XXIV) argue that classifier systems originally proposed by Holland for rule discovery using genetic algorithms offer a basis for integration of the styles of representation and processing used in both SAI and NANN systems. Honavar (chapter XXV) critically examines a broad range of computational models of learning drawn from statistical pattern recognition and NANN, syntactic pattern recognition and SAI, as well as evolutionary methods such as genetic algorithms. It also identifies a number of areas for potentially beneficial cross-fertilization among the various paradigms.

7 Summary

At first glance, traditional artificial intelligence systems and artificial neural networks appear to be radically, if not fundamentally different. However, both the paradigms for modelling intelligence share common origins as well as the same fundamental working hypothesis - that cognition can be modeled by computation. They both rely on different but essentially equivalent models of computation. This guarantees that any task that can be performed by an artificial neural network can be performed in principle by a suitably designed artificial intelligence program and vice versa. But as nearly five decades of collective experience of both theoreticians and practitioners in several areas of computer science - e.g., programming languages, computer architectures, and algorithms - has demonstrated, the important questions have to do with efficiency, robustness and elegance and the various tradeoffs among different languages, computers and algorithms conditioned by such choices.

Both traditional artificial intelligence systems and artificial neural networks have some obvious (and some not so obvious) advantages as well as disadvantages for certain types of problems under certain types of performance and design constraints. Together, they span a large and interesting space of possible designs for intelligent systems. Examples of hybrid systems resulting from judicious integration of concepts, constructs, techniques and technologies drawn from both traditional artificial intelligence systems (and related paradigms such as syntactic pattern recognition) on the one hand and artificial neural networks (as well as related paradigms such as statistical pattern recognition and fuzzy logic) on the other demonstrate the potential benefits of such an approach to the design of truly robust, flexible, general, efficient, and adaptive intelligent systems. The rather severe practical limitations of today's artificial intelligence and neural network systems strongly argues for the need for a systematic exploration of the space of such hybrid designs that incorporate strengths of both. This book is dedicated to pulling both traditional artificial intelligence and artificial neural network approaches closer, if not back together.